2018年度　修士論文

仮想マシンの安定度に応じた動的マージン決定およびそれを用いたライブマイグレーション手法に関する研究

2019年　2月　1日(金)提出

早稲田大学大学院　基幹理工学研究科
情報理工・情報通信専攻　深澤研究室

学籍番号 :5117F037-6
坂本竜

指導：深澤　良彰　教授
指導研究名：ソフトウェア開発工学研究

目 次

第1章　はじめに

コンピューターが生まれて以来, インターネットやハードウェアの発達に伴い分散コンピューティングなどの「コンピュータリソースをネットワークを経由したサービスとして提供する」技術が注目されるようになった. 2000 年代初頭にはクラウドコンピューティング（以下, クラウド）と呼ばれる新たなコンピューティングの形態が出現した. クラウドではグリッドコンピューティング等と同様にネットワーク越しにコンピューターリソースを利用でき, クラウドのコンピューティングリソースがどこにあるのかを一切意識せずに利用することができる. 例えば利用者からは 1 つの仮想マシンを利用しているだけに見えても,CPU・RAM は日本のデータセンター,SSD などのストレージはアメリカのデータセンターにあるということがある. クラウドではコンピューターリソースはサービス提供業者側で管理・運用されるため PC などのクライアントと利用料金だけがあれば最低限利用可能である.

1.1　クラウドコンピューティングとは

本節ではクラウドについてより詳しく説明し, クラウドの分類や用いられる技術などについて具体例を交えて述べる. クラウドとは, インターネットを通じてクラウドサービスプラットフォームからコンピューティング, ストレージ, データベースなどのさまざまな IT リソースを利用することができるサービスの総称を指す.

「クラウドコンピューティング」という言葉を最初に使用したのは,2006 年の Search Engine Strategies Conference において Google CEO のエリック・シュミットがクラウドの概念について語った時 [6] だとされている. インターネット（雲）の「向こう側」にあるさまざまな膨大なリソースを物理的な場所を意識せずに利用できる.（図 1.1）

1960 年代にはタイムシェアリングシステムなどに代表されるデータセンターの利用サービス,1990 年ごろには SaaS の元となる ASP(Application

Service Provider) などが存在し,「ネットワークを通じてコンピューティングリソースを利用する」という形態は新しいものではない.

しかし,2000 年代から発展しているクラウドは世界中のユーザーがサーバーの場所を意識することなく利用できる点で広く受け入れられている. 分散コンピューティングの時代には専門の知識や技術の習得が必要で一般的な利用は難しかったものがクラウドでは簡単になった.

1.2　クラウドの分類

1.2.1　配置形態による分類

プライベートクラウド

企業等がクラウド技術を利用した環境を自社内に構築することでイントラネットを経由して利用できるようにした形態である. インターネットに接続されないためセキュリティの観点で優れている. また, マシンスペックなどを自由に決定できるため要求に合わせた環境構築が可能になる.

パブリッククラウド

インターネット経由でクラウドサービスを提供する形態である. 一般的に言われる狭義のクラウドコンピューティングはパブリッククラウドを指すが, 用いられる技術はプライベートクラウドと変わらない. 初期費用がかからず即時導入可能で, インフラの運用などをサービスプロバイダーに任せることができ, 手間がかからない.

1.2.2　提供サービスによる分類

クラウドは, システムの構成要素のどこまでをサービスとして提供するかによって SaaS,PaaS,IaaS に分離できる.(図 1.2) 構成要素はネットワーク, ハードウェア,OS, ミドルウェア, アプリケーションの 5 つに分けられる.[15]

SaaS

SaaS（Software as a Service）とは, メールやファイル共有サービスのような従来はパッケージ製品として提供されていたソフトウェアをインターネットを通じて利用できる形態を指す. 例としてはメールや表計算などさまざまなアプリを提供する「Google Apps」やファイル共有ストレージサービス「Dropbox」などがある.

PaaS

PaaS（Platform as a Service）とは, アプリケーションが動作するための OS やハードウェア, ミドルウェアなどをまとめたプラットフォームを提供するサービスを指す. 利用者は Web アプリケーションを開発するエンジニアであることが多く, アプリケーション（ソースコード）をビルド・デプロイして Web サービスを提供できる「Google App Engine」や「Heroku」などがある.

IaaS

IaaS（Infrastructure as a Service）とは, 利用者が自由に環境構築をできる仮想マシンをはじめとしたネットワークなどのインフラを提供するサービスを指す. 「Google Compute Engine」や「Amazon EC2」などがある. PaaS と比較して利用者が行うことが多い反面自由度が高くなっている.

本研究では, クラウドの中でも IaaS に焦点を当てる. IaaS では事実上無限と言える数の仮想マシンを利用することができ, 必要に応じた数の増減も柔軟に行える. さらに,SaaS,PaaS と比べて低レイヤーであるため利用者の要求に応じて OS から自由にカスタマイズ可能である. クラウド全体に言えることではあるが, 料金も使用する規模・期間に応じて課金されるため, 必要な時に必要な分だけ利用可能である. IaaS は Web サーバーとして利用されることが多く,Web サーバーの特徴として外部からの大量のリクエストを処理することがあげられる. それによって Web サーバーの負荷は一定ではなく, 更に Web サーバーでどのようなサービスを提供しているかによっても変化する. クラウドであれば仮想マシンの追加・削除が容易に可能であるため, リクエストの増減にも柔軟に対応することができる.

1.3　仮想化

クラウドに用いられる主要な技術に仮想化技術がある．仮想化とはCPU,RAM などのハードウェアリソースを, 物理的な構成に関係なく論理的な統合や分割を可能にする技術のことである．仮想化技術はネットワークやストレージなどにも適用できる．ここではそれらの仮想化技術を用いた仮想マシンで構成されるサーバー仮想化ついて述べる．図 1.3 は仮想マシンの構成を示す．

仮想マシンには OS 仮想化とハードウェア仮想化の二種類が存在するが, 本手法には深く関わらないためここではそれぞれについての説明は割愛する．仮想化によって一台の物理サーバーを複数台の論理的サーバーに分割することができ, それぞれで独立して OS やアプリケーションを起動可能である．仮想化自体はクラウドだけではなくオンプレミスでも利用されているが, クラウドにおいては仮想化なしには実現することは困難であった．仮想化のメリットとしてはリソースの有効活用, コスト削減, 柔軟なスケール変更などがあげられます [12]．リソースの有効活用はクラウド最大のメリットとも言え, コスト削減にも繋がるものである．仮想化を利用せずに 1 台の物理サーバー上で 1 つのアプリケーションを動作させる場合, サーバーの種類によって稼働率や負荷が異なる．その上サーバーのスペックは余裕を持って選択されることが多く,「余剰設備」が発生する．

これに対して仮想化では必要に応じてリソースを割り当てることができるため, 負荷が大きいと予想される場合にはサーバーを即座にスケールアップすることができる．そうすることで利用者はサーバーの負荷に合わせて効率的にリソースを利用することが可能になる．また,「余剰設備」に関しても複数のサーバー全体として余剰を考慮すればいいためさらに効率が高くなると言える．「余剰設備」が少なく済むため設備にかかる導入コスト, 運用コストを削減することができる．また, 物理サーバーの台数を削減できるためメンテナンスにかかる手間も削減可能である．

仮想化をすることでリソースの分配の自由度も跳ね上がる．あるサーバーに突然リクエストが集中し性能を一時的に向上させる（スケールアップ・スケールアウト）必要があると言うような場合も即座に対応可能である．そのあとすぐに元の性能に戻すことも当然可能である．また, 仮想マシンの構成は物理マシンの構成に縛られないため利用者が必要としている CPU,RAM, ストレージなどそれぞれの性能を別々に自由に調整することが可能であるため, 需要と供給のミスマッチも起こりにくい．

1.4　クラウドにおいて考慮される点

本節ではクラウドで考慮される点について, 特にクラウド提供者の視点から述べる.

提供側はデータセンターを用意して複数の物理マシンを運用し, その上で仮想マシンを起動してサービスを提供している. 仮想マシンは利用者からの要求によって頻繁に増減し, それぞれの仮想マシンの規模や負荷も様々である. 多くの物理マシンを運用するため, 電気代や機器代などの設備投資費用が多くかかっている. これを減らすことができれば利益を大きくできたり提供料金を安く抑えることが可能になる.

提供者は基本的にクラウドサービスを提供することで利益を得ている. 提供するクラウドをより多くの利用者に使ってもらうためには価格や信頼性などの QoS（Quality of Service）, サービスの種類などで他の提供者を上回ることが目標となることが考えられる. したがって価格や QoS はクラウド提供者の考慮する点であると言える. クラウドの利点はコストを支払えば容易にリソースを用いることができる点であるから, かかるコストや信頼性が変わらなければ利用者視点ではどのサービスも同一に見えてしまう.

QoS とサービスの提供価格は相反する要素であると言える. QoS を向上させるためにより多くの設備投資を行えばそのコストは価格に反映されることとなってしまうからである. クラウド提供者は設備コストを抑えるために物理マシンの台数を減らしたりスリープ状態の物理マシンを増やしたりしたいため, ライブマイグレーション技術を用いたりオーバーコミットを行ってリソースをより効率的に利用する. ライブマイグレーションとオーバーコミットについては次章で詳しく説明することとする.

本論文の残りの構成を以下に示す. 第 2 章ではオーバーコミットとライブマイグレーションについて説明し, 既存技術の問題点・本研究の目的を述べる. 第 3 章では本研究の関連研究について説明する. 第 4 章ではライブマイグレーションにおける問題点を解消し, 仮想マシンを容易に移動できるという利点を活用し, より効率的に物理リソースを利用できる手法を提案する.. 第 5 章では提案手法のシミュレーションによる実験・結果を述べ, それに対して考察を述べる. 第 6 章では本論文をまとめ, 本研究の将来的な課題を述べる.

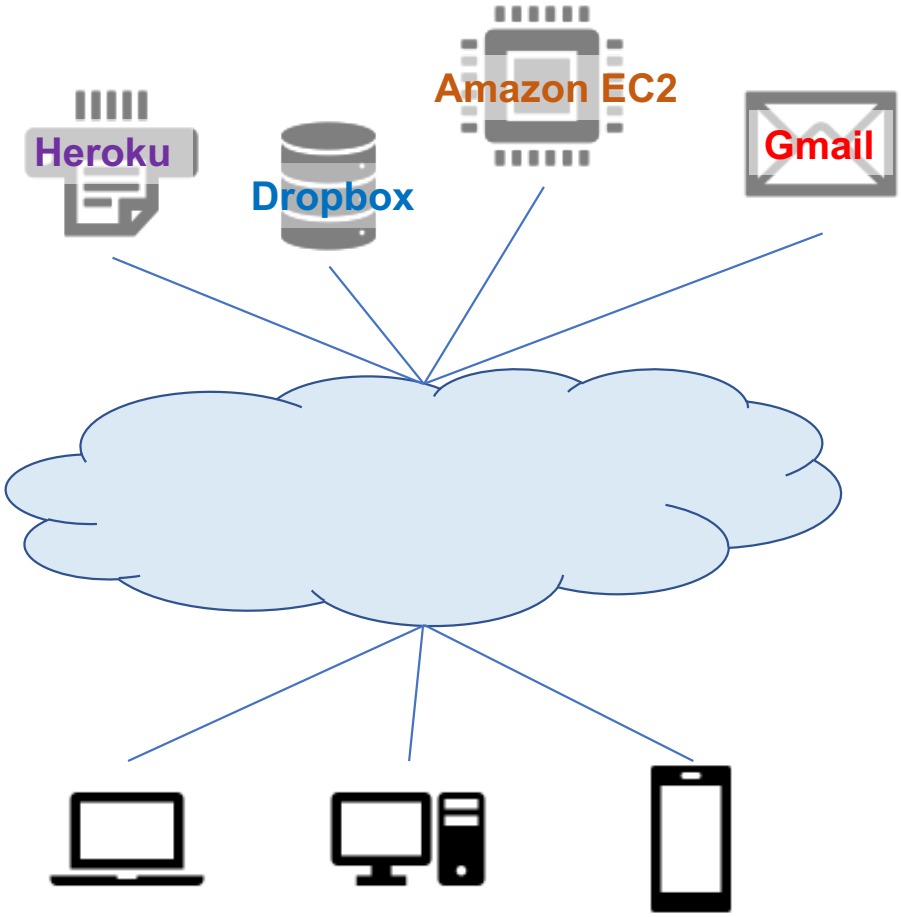

図 1.1: クラウドのイメージ

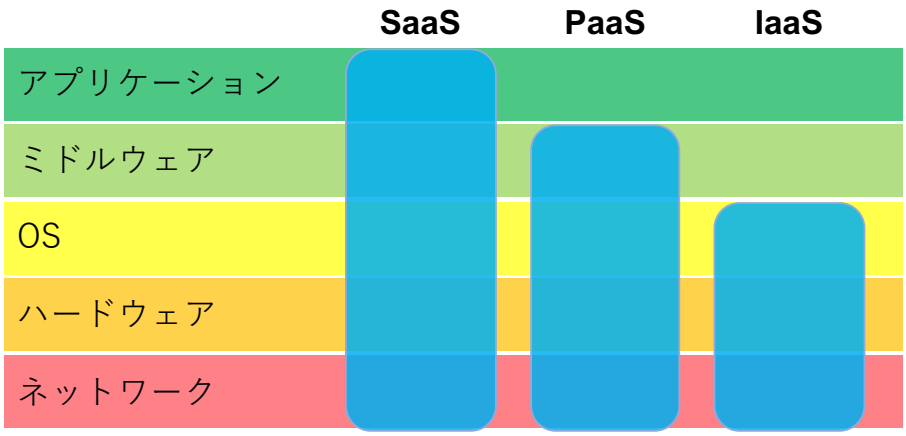

図 1.2: SaaS,PaaS,IaaS の違い

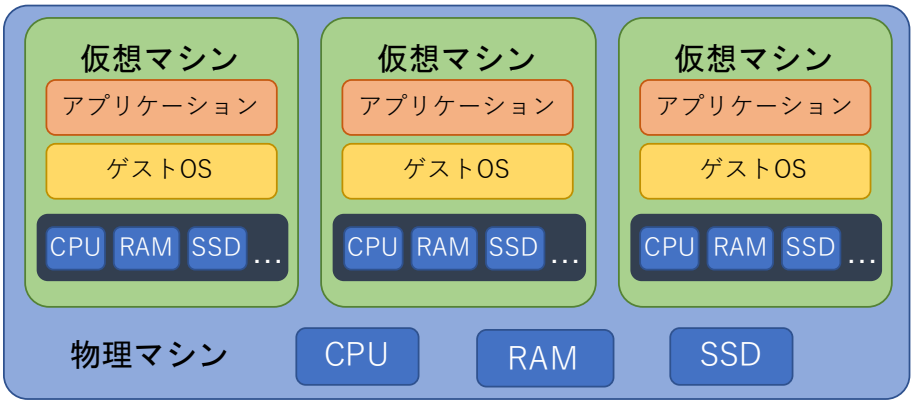

図 1.3: 仮想マシンの構成図

第2章　オーバーコミットとライブマイグレーションおよび既存技術の問題点

本章では, 本研究と深く関わってくる CPU のオーバーコミットとライブマイグレーションおよびそれらにおける既存技術の問題点について述べる.

2.1　オーバーコミット

CPU のオーバーコミットは, 物理マシンが持っている CPU のコア数を仮想マシンに割り当てられている仮想 CPU の合計数が上回っている状態を指す. 例えば 10 コアを持つ物理マシンに 2 つの仮想 CPU を持つ仮想マシンを 6 つ配置する場合,2 コア分オーバーコミットされている状態である. これは CPU は常に 100%の性能を使い続けることは通常では考えにくいからである. また, 一つの仮想 CPU の動作周波数を物理 CPU の動作周波数より小さく設定すればさらに多くの CPU 数で稼働させることが可能である.

例えばホストマシンが 3GHz6 コアの CPU を持っている場合,3GHz1 コアの仮想マシンを 6 台稼働させることができる. また 1GHz1 コアの仮想マシンであれば 18 台を稼働させることも可能で, 更には CPU は先に述べたとおり 100%を常に使うわけではないため,CPU 使用率が 50%の仮想マシンであれば（現実的には余裕を持たせるが）さらに倍の台数を稼働させることが可能である. 中小企業や研究室などでの小規模なプライベートクラウドではオーバーコミットを行うことによって物理マシンの台数を削減することが可能になり, 限られたリソース・予算で運用するために有効である. また, パブリッククラウドの Amazon の AWS の t2 インスタンスや Microsoft の Azure の A0 インスタンスでもオーバーコミットは行

われていて, 提供料金が低くなっている.

オーバーコミットの利点は先に述べたとおり一つの物理マシンにより多くの仮想マシンを配置することができる点である. しかし当然オーバーコミットにも欠点があり, 仮想 CPU の数を増やしすぎることでリソースの待ちが起こりやすくなったり, 仮想マシンが消費するリソースの総和が物理リソースを超えてしまった場合に全ての仮想マシンのパフォーマンスが低下してしまう点である. 先に述べた例であれば, 仮想マシンの負荷が上昇して 80%になってしまった場合, ホストマシン全体の負荷が 160%となってホストマシンの性能を上回ってしまい, パフォーマンス低下が起こってしまう. オーバーコミットによる多少のパフォーマンス低下の可能性を許容できる場合はオーバーコミットの利用価値は大きいと言え, 実際に AWS の t2 インスタンスなどのサービスがある通り, 現実にそういった要求は多く存在することがわかる.

2.2　ライブマイグレーション

前節で述べたオーバーコミットの欠点を補うためにライブマイグレーションという技術を用いることができる. ライブマイグレーションはホストマシン上の仮想マシンを OS やアプリケーションをシャットダウンすることなく他のホストマシンへ移動させることを可能にする技術である. (図 2.1) ある仮想マシンが高負荷になった・なりそうな際にも余裕のあるサーバーに移動させることで性能の低下などの影響を最小限に抑えることが可能である. ライブマイグレーションを用いると仮想マシンの利用者は仮想マシンの移動を意識することなくクラウドサービスを利用でき, 仮想化による恩恵を最大限受け柔軟な運用が可能になる. ライブマイグレーションはオーバーコミットと組み合わせないとしても有用な技術であり, ホストマシンの OS やメンテナンスの際などにも一時的に他のホストマシンへ退避させることで容易になる. 実際のアプリケーションとしては VMWare の提供する vSphere や Microsoft が提供する HyperV などで実装されており, クラウドを運用する上で重要な技術の一つとなっている.

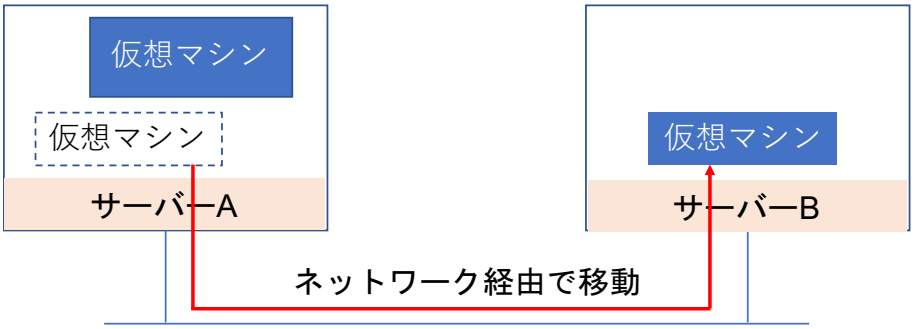

図 2.1: ライブマイグレーションのイメージ

2.3　負荷予測

ライブマイグレーション技術は負荷分散などリソースの効率的な管理にとても有用である. しかし, 仮想マシンが高負荷になったことを検知してからライブマイグレーションを行うと, ライブマイグレーション自体にも時間ががかる上オーバーヘッドが発生してしまい, さらにホストマシンの負荷を高めることになってしまう可能性もある. これを防ぐためには事前に仮想マシンが高負荷になることを何かしらの方法で予測・検知する必要がある. その方法として負荷予測があり, リソースとして核をなす CPU 使用率の負荷予測が用いられる場合が多い [?][4]. CPU の負荷予測をすることによってどの物理マシンの負荷が大きくなりそうかを検知することが可能になり, それを基に予防的に仮想マシンをマイグレーションしておくことで前もって性能低下に対処することができる. さらには負荷が低すぎる物理マシンを検知することにより, そこに配置されている仮想マシンを全て他の物理マシンにマイグレーションしてその物理マシンをスリープ状態にすることもできる. これによって電力消費なども抑えることも可能になる.

2.4　既存技術における問題点

既存の研究において CPU 使用率の負荷予測を用いたライブマイグレーションを行い, 負荷が高まってからライブマイグレーションを行う方法に対してホストマシンが過負荷になってしまう回数やマイグレーションの回数, リソースの使用量を減らす手法は存在する [4]. CPU の負荷予測を

するにあたり考慮すべきこととしてCPU使用率は変動が大きい点があり, 正確に予測することは難しい. CPU使用率は1分や10分程度のスケールで見ても, 数秒という小さなスケールで見ても変動が大きい. これは, 動作しているアプリケーションにはネットワークやファイルの読み書きの待ちなどがあり, 常にCPUの性能を一定に使ってはいないからである.

既存手法においては, 予測が正確に行えない問題点に対して, 予測値に対してある程度（数%〜十数%程度）のマージンを取ることでこれを吸収しようとしている. しかしながら,CPU使用率の変動がアプリケーションによるものであるから変動自体の特性もアプリケーションによって異なる [1]. 図2.2〜2.4はHP研究所が発表したアプリケーションによるCPU使用率の変化特性についての報告書に掲載されているCPU使用率の特性を表した図であるである [1]. 3つのグラフは全て異なる見た目をしており, 図2.2は常に大きく変動をしていて, 図2.3は全体として大きく変動はしているが, 小さいスケールで見ると図2.2と比較して小さな変動である. 図2.4は基本的に低いCPU使用率を示しているが, スパイク状の上昇が認められる. このようにCPU使用率の変動の特性はアプリケーションによっても異なるため, どの予測値に対しても一定のマージンを取ることは最適ではないと言える.

2.5　本論文の目的

節2.4で述べたように,CPU使用率予測へのマージン設定をより適切に行うにはどのような変動特性を持っているかを考慮することが必要と言える. そのため, 本研究ではより適切なマージン設定を行い効率的なリソース管理を行うことを目的とし, 従来の手法で用いられた固定マージンに収まらないようなCPU使用率の変動やあらゆる仮想マシンに一律にマージンを付与することによるリソースの無駄に対処する. 本論文ではCPU使用率の変動特性として予測値と実際の値がどれだけずれているかを数値化し,「安定度」と呼ぶ. 安定度は時間と共に動的に変化し, それを基にマージンを動的に決定する手法および, 安定度を考慮したマージンを含めたCPU使用率を基にライブマイグレーションを行うエージェントを提案する.

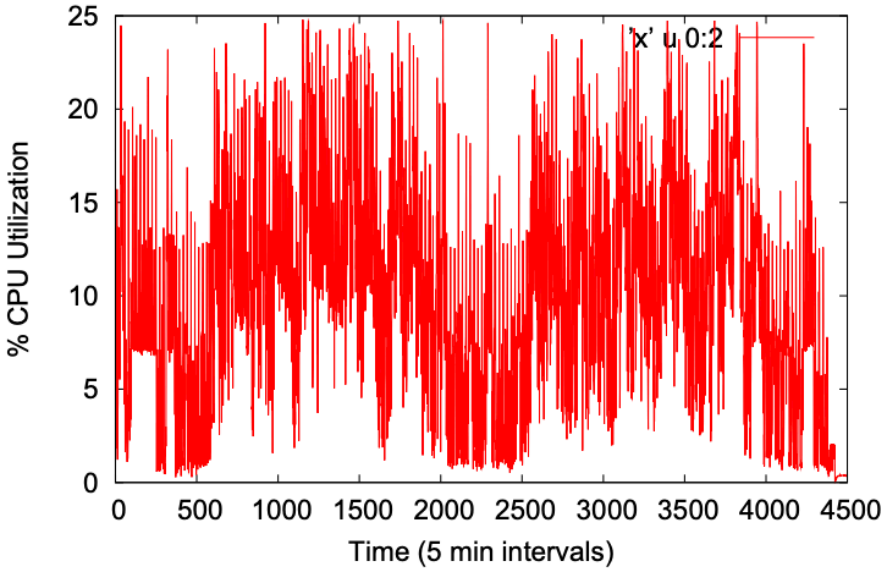

図 2.2: CPU 使用率の変化特性 1[1]

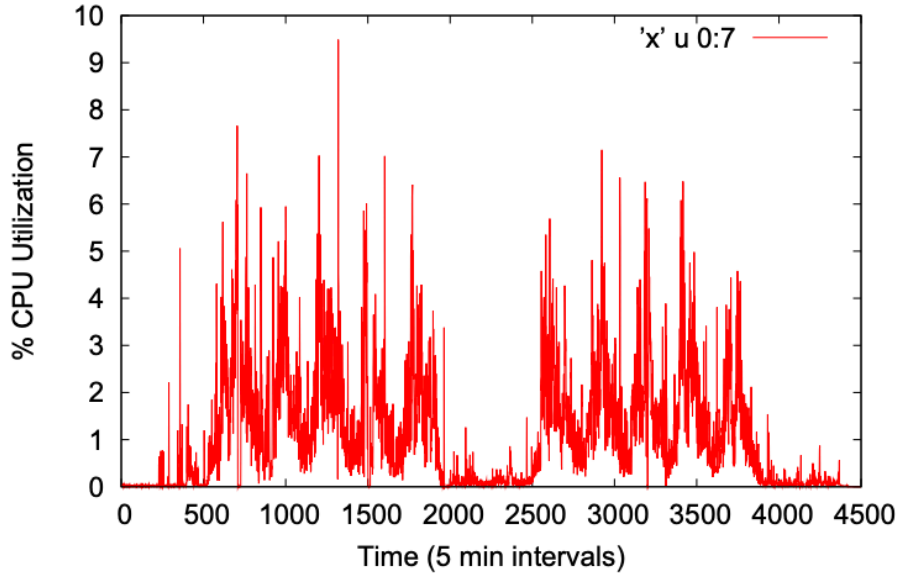

図 2.3: CPU 使用率の変化特性 2[1]

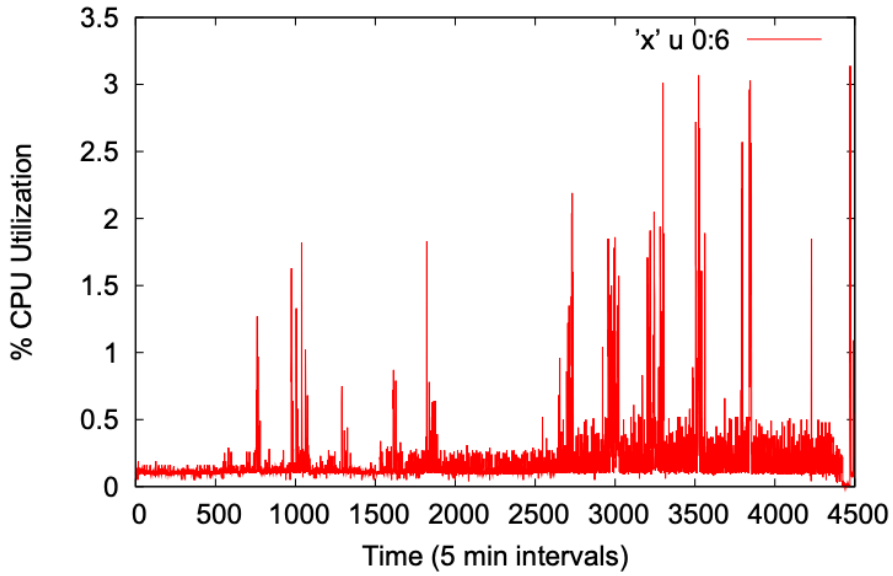

図 2.4: CPU 使用率の変化特性 3[1]

第3章　関連研究

3.1　負荷予測に関する研究

負荷予測にはCPU使用率を用いるものの他に, ネットワークリソースやメモリリソースの使用率を用いるものもある. [4][7] また, 負荷予測自体のアルゴリズムには一次関数に線形回帰 [4] を用いて近似する手法などがある. また,Suhib Bani Melhem ら [11] はマルコフモデルを用いて負荷の状態変化を予測・検知しており, いつライブマイグレーションするかを決定している. M. Duggan らはCPU使用率をニューラルネットワークを利用することでより正確に予測できる手法を提案している [?]. 更にすぐ将来の値だけでなく, その先も同時に予測している. PaaS において最適なオートスケーリングを行うために機械学習を用いてQoSの予測をする研究も行われている [?].

3.2　ライブマイグレーションに関する研究

仮想マシンの配置を決定する管理エージェントに用いられるライブマイグレーション技術は利用目的がいくつかあり, それによりエージェントの方針が異なってくる. Ghribi ら [5] は, ライブマイグレーションを用いて休止状態にできるホストマシンを増やし, データセンターの電力消費を削減することに用いている. また, 秋山ら [2] や Vincenzo De Maio ら [10] はマイグレーション自体にかかる電力コストも含めた適切なライブマイグレーションを提案している. Zhaohui Wu ら [14] はCPUの温度を予測してデータセンターの熱管理に利用しており, それによって冷却にかかる電力消費を削減している. 従来ではマイグレーションが動的に行われる状況には利用できなかった手法を利用可能にしている.

Jananta Permata Putra ら [13] はライブマイグレーションをロードバランシングに用いている. ライブマイグレーションを用いない場合は実行時間がマイグレーション元のサイズに比例して大きくなっていたが, ライブ

マイグレーションを用いることでサイズに関係なく新しい仮想マシンを立ち上げることに成功している. VMWare は閾値を設定し, 閾値を超えたらマイグレーションを行う手法を用いている [?].

ライブマイグレーション自体を効率化する研究もなされており,C.Li ら [9] ライブマイグレーションにかかる伝送時間を短縮するためにネットワーク帯域幅に応じて適切な圧縮アプローチを選択することで効率化を図っている. M. Dabbagh らは効率的なライブマイグレーションを行い回数と物理マシンの数を減らすことで電力消費を削減する手法を提案している [?].

第4章　安定度に応じたマージン決定およびライブマイグレーション手法の提案

本研究では, クラウドにおける物理リソース管理の効率を高めることが目的である. 既存の技術では一定のマージンを付与するにとどまり, 紹介した報告書 [1] で述べられているように CPU 使用率の変動はアプリケーションごとに異なる特性を持っており, 適切な対処ができているとは言い難い. そこで本論文では, 物理マシンで稼働する仮想マシンの負荷の特性に応じて動的に付与するマージンを変動させ, それに応じてマイグレーションする仮想マシン及びそのマイグレーション先を決定する機構を提案する. 本研究では研究室や中小企業などが保有する, オーバーコミットが必要になる物理リソースが限られている小規模なプライベートクラウド環境を想定する. ここで, 本論文で提案する機構をリソース管理エージェントと呼ぶこととする. リソース管理エージェントによって各々の仮想マシンの状態に応じた効率的なリソース管理を実現する. 本章ではリソース管理エージェントの構成およびその機能について説明する. また, 更なる提案としてリソース管理エージェントにおけるオーバーコミットの度合いの動的な変更についても説明する.

4.1　リソース管理エージェントの構成

本論文で提案するリソース管理エージェントの機能は次の 4 つに分かれていて,2 つ目以降が本論文の提案にあたる.

- 負荷を予測する機能
- 予測値を用いて仮想マシンの安定度を導出する機能
- 導出された安定度を基にマージンを決定する機能

- マージンを含む予測値と各仮想マシンの安定度を基にマイグレーションを決定する機能

また, リソース管理エージェントを利用する前提として過去の CPU 使用率のデータを利用して将来の負荷の予測値を算出する予測機能が必要となる. 図 4.1 にリソース管理エージェントの概要図を示す. CPU 使用率の過去データを保存したデータベースから最新の一定時間の CPU 使用率を取り出し, それを用いて負荷予測及び安定度の計算を行う. 導出された安定度から各仮想マシンのマージンを決定しマイグレーションを行う.

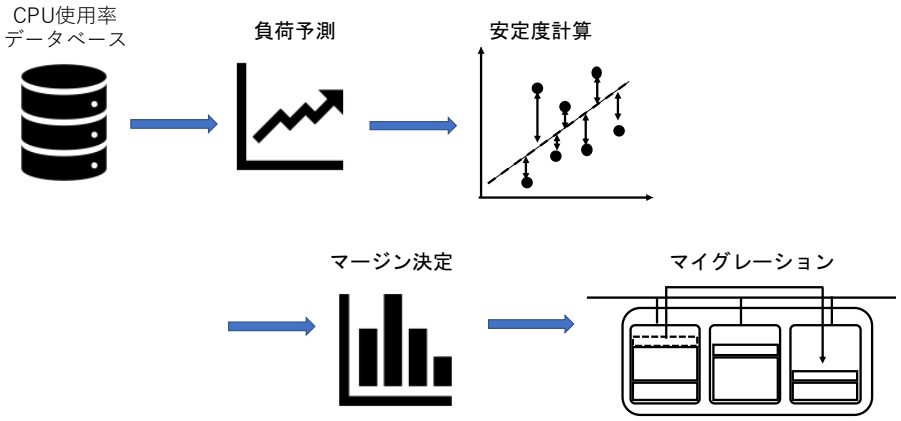

図 4.1: エージェントの構成

4.2　負荷を予測する機能

本節ではリソース管理エージェントを利用するために必要な負荷予測機能について説明する. 各ホストマシンの負荷状況を把握し過負荷になる前にライブマイグレーションを行うためには負荷予測が必要である. 本手法では過去の CPU 使用率を利用して関数を導出し, それを用いて将来の負荷を予測する. 入力として CPU 使用率を一定間隔ごとに収集したデータを用い, 出力は線形回帰を用いて次の時刻の CPU 使用率および線形回帰の結果得られた予測関数である一次関数となる. 線形回帰には既存研究でも CPU 使用率を予測する手段として用いられている [3], 残差の二乗和が最小となるように係数を決定する方法である最小二乗法を用いる. 線

形回帰に最小二乗法を用いた場合, モデル関数は一次関数であり,

$$f(x) = ax + b \tag{4.1}$$

とおくと,a と b は次の式 4.2,4.3 のように求められる. 最小二乗法では誤差の二乗和を最小にすることで最も確からしい関数を求めている.

$$a = \frac{n\sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k \sum_{k=1}^{n} y_k}{n\sum_{k=1}^{n} {x_k}^2 - \left(\sum_{k=1}^{n} x_k\right)^2} \tag{4.2}$$

$$b = \frac{\sum_{k=1}^{n} x_k^2 \sum_{k=1}^{n} y_k - \sum_{k=1}^{n} x_k y_k \sum_{k=1}^{n} x_k}{n\sum_{k=1}^{n} x_k^2 - \left(\sum_{k=1}^{n} x_k)\right)^2} \tag{4.3}$$

4.3　安定度を導出する機能

本節では最小二乗法を用いて予測された CPU 使用率と過去データを利用して仮想マシンの安定度を導出する. 予測された負荷に対して動的なマージンを適切に設定するためにはまずどれほどマージンが必要かを表す指標である安定度を導出する必要がある. この機能は予測関数と過去データを入力とし, 安定度を出力とする. 安定度は予測関数と過去データの各時間ごとの誤差 (%) を算出し, その総和の標準偏差を計算し,-1 を乗算することで求められる. 安定度は仮想マシンの CPU 使用率の変動がどれほど安定しているかを示し, 不安定なほど負に大きくなり, 安定しているほど 0 に近くなる. 例えば以下のように 35%と 45%を上下しているような CPU 使用率だとすると, 最小二乗法によって導出される予測関数は

$$y = 40$$

であり, 安定度は-5 となる. 言い換えると安定度は CPU 使用率の予測とどれくらいの誤差が生じている可能性があるのかを示しているとも言える.

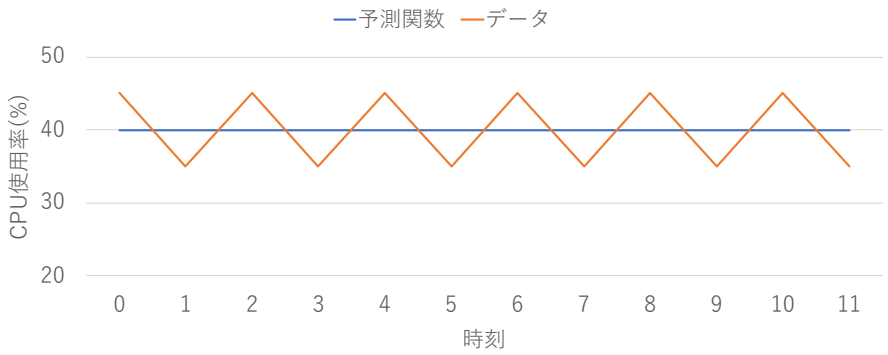

図 4.2: 安定度の例

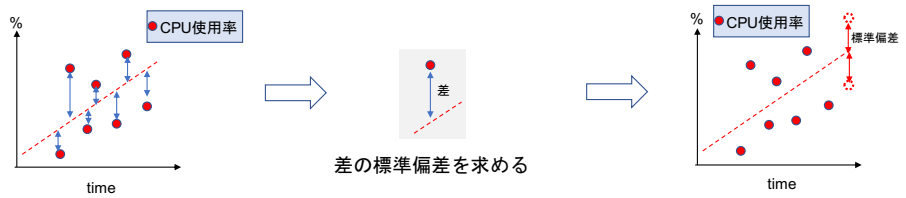

図 4.3: 安定度の導出のイメージ

4.4　マージンを決定する機能

本節では節 4.3 で求められた安定度を基にした仮想マシンのマージンの決定を説明する．この機能は固定値では安定度が低い状態では不足し、安定度が高いと過剰になるマージンを適切な大きさに設定する．安定度は節 4.2 で求めた一次関数からどれだけずれているかを示すものであるが, 仮想マシン上でアプリケーションが動いておらず CPU 使用率が 0%のままだとほぼ 0 となり小さすぎるまた一方で大きすぎる場合も考えられる．実際の CPU 使用率が 30%であるのにマージンが 40%で合計で 70%分のリソースを確保してしまうと過剰である．本論文がマージンを動的に設定する目的は仮想マシンに対して最適なサイズのマージンを設定することであるから、過剰であったり過小であることは不適切である。そのため本論文では安定度に係数をかけてマージンとすることで, マージンに上限と下限を設定して適切な範囲内に収める．本研究の途中経過である研究 [16] によって求められた最適なマージンの係数設定の式は次の 4.4 となっ

ている.

$$a = \begin{cases} 4 & (Std\_Dev < 1) \\ \frac{77}{18} - \frac{5}{18} Std\_Dev & (1 < Std\_Dev < 10) \\ 1.5 & (Std\_Dev > 10) \end{cases} \tag{4.4}$$

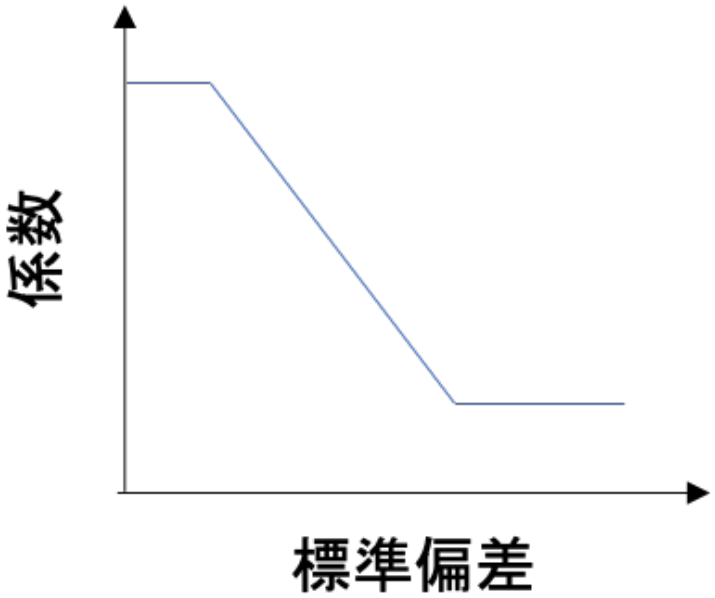

図 4.4: マージンの係数のイメージ

4.5　安定度を基にマイグレーションを決定する機能

本節では節 4.4 で決定されたマージンを基にしてホストマシンの過負荷を検知し, マイグレーションを行うエージェントの機能について説明する. この機能は過負荷になると予測された仮想マシンを適切なホストマシンに移行させる. マージン入り予測値を用いて計算することで自動的に安定度の低い仮想マシンが多いホストマシンはマージンが大きく, そうでないホストマシンはマージンが小さくなるような性質を持つ. 本機能は過負荷検知と仮想マシンの再配置の二つに別れており, それぞれについて説明する.

4.5.1　過負荷予測・検知

ホストマシンの CPU 使用率が高くなると, 過負荷を防ぐために配置されている仮想マシンを移動させる必要がある. 本エージェントは, マージンを含めた各仮想マシンの負荷予測の合計が, ホストマシンの性能を超える場合にマイグレーションを行う必要があると判断する. マイグレーションが必要とされたホストマシンは配置されている仮想マシンを負荷が高い順に並べ, 残りの負荷合計がホストマシン性能に収まるまで仮想マシンをリストに加え続ける. これを全てのホストに対して行い, 移行する必要のある仮想マシンリストが生成される. 過負荷予測・検知アルゴリズムを次の 1 に表す.

Algorithm 1 Detection

Require: $predictedvalueoftheloadofhostmachines$
$currentloadofeachvirtualmachines$
Ensure: $moveList : alisttobemigrated$
for $i = 1$ to $length\ of\ hosts$ **do**
if $usage_of_hosts_i < 100\%$ **then**
for $j = 1$ to $length$ of $hosts_i.vms$ **do**
$moveList \leftarrow hosts_i.vms_j$
if $usage_of_hosts_i < 100\%$ **then**
break
end if
end for
end if
end for

4.5.2　仮想マシンの再配置

次に, 作成された移行する仮想マシンリストの移行先を決定する. まず仮想マシンリストに入っている仮想マシンは消費するリソースが大きい順にソートされる. そしてホストマシンはリソースの余裕が大きい順にソートされる. ホストマシンの余裕はホストマシンの性能から使用されているリソースと安定度を差し引いたものであり, 次の式 4.5 で表される. *capacity of host* はホストマシンのサイズを表しており、*total usage of host* はホストマシンで消費されているリソース、*total stability of host* はホストマシンに配置されている仮想マシンの安定度の合計である。

$$\begin{aligned} rest &= capacity\ of\ host \\ &- total\ usage\ of\ host \\ &- total\ stability\ of\ host \end{aligned} \tag{4.5}$$

次に仮想マシンのリストから一つ仮想マシンを取り出し, 最も CPU 使用率に余裕のあるホストマシンに配置する. その後ホストマシンを再びソートし, 仮想マシンのリストが空になるまでこれを繰り返す. どのホストにも配置できない仮想マシンが存在する場合, 新しく空のホストマシンを追加する. 仮想マシンの再配置アルゴリズムを 2 に示す.

Algorithm 2 Reallocate

Require: $alistofhostmachines$, $alistofmigrationlist$
Ensure: $newHosts$
  for i = 1 to length of vms **do**
    $sorthosts$ ($orderbydescending$)
    $LargestHost \leftarrow hosts_0$ ($largestsurplus$)
    if $vms_i.usage > LargestHost$ **then**
      $hosts \leftarrow emptyHost$
    else
      $LargestHost \leftarrow vms_i$
    end if
  end for

オーバーコミットの度合い

本論文における提案では主に固定値のマージンでは不足してしまう安定度が低い状況に対処するためのものである. そこで, 更にリソースの利用効率を高めるために, ホストマシンに配置されている仮想マシンの数に応じてオーバーコミットの度合いを高める手法も提案する. 本手法では基本的にはホストマシンの容量に空きがなくなるまで仮想マシンを配置し続けるが, 更にリソースを効率的に利用するために更なる提案としてオーバーコミットの度合いの動的な変更について本節では説明する. ホストマシンに配置されている仮想マシンの数が多い場合は更にオーバーコミットをしてより多くの仮想マシンを配置するが, これをモデル化して説明する. ここではホストマシンのリソースを仮想マシンのリソース利用の合計が超えてしまった状態を「違反」, 各仮想マシンの CPU 使用率の予測値とマージンを合わせた値を「マージン入り予測値」と定義する. まず次のように条件を定義する

- 仮想マシンの数: n
- ホストマシンの容量: C
- 仮想マシンのサイズ: c
- 違反した際に超過する CPU 使用率: 5%
- 仮想マシンの違反確率: p

- マージン: s%（簡単のため一定とする）

この条件において, 次のことが導かれる.

- マージン: $c \times s$
- ホストマシンの未使用リソース: $c \times s \times n$
- 一回の違反が余分に消費するリソース: $5 \times c$
- ホストマシンのリソースが全て消費される確率:
  $\frac{c \times s \times n}{5 \times c} = \frac{s \times n}{5}$ 個の仮想マシンが同時に違反する確率
  $= p^{\frac{s \times n}{5}}$

以上から, 配置されている仮想マシンがホストマシンのリソースを全て使い切ってしまう確率は n が大きいほど低いことがわかる. つまりホストマシンに配置されている仮想マシンの数が多い場合はオーバーコミットの度合いを上げても違反しにくいことが言える.

第5章　実験と考察

本論文の提案している手法の有用性を評価するにあたり評価実験およびその考察を行った. 本章では実験に際して行った実装も含めて説明する.

5.1　実験の方針

本論文の提案手法の目的は限られたリソース環境におけるリソース利用効率の向上であり, リソース管理エージェントを用いて仮想マシンごとのマージンを動的に決定する手法を提案することにより実現しようとしている. 本実験では節 4.5.2 において定義した「違反」,「マージン入り予測値」に加えて, ホストマシンの数を「ホスト数」,1 つの仮想マシンを他のホストマシンに移動させることを「VM マイグレーション」と呼ぶことにする.

本実験では既存のマージンが固定値の手法と比較することとし, 評価基準として「違反回数」,「マージン入り予測値の合計」,「VM マイグレーション回数」,「ホスト数」を用いる. 違反回数はマージンが不足してしまう状況が減少したかどうか, マージン入り予測値の合計はリソースの確保量が従来手法と同程度か, マイグレーション回数は配置するホストマシンをより適切に選択できているか, ホスト数は全体としてハードウェアにかかるリソースが増加していないかをそれぞれ検証する指標として設定している. これらは本論文の提案を適切に評価する上で必要と考えられる. また, 既存の研究でも違反回数, マイグレーション回数は指標として利用されている.

CPU 使用率のデータについては多くの既存の論文でも評価に用いられている [?][4]CoMon プロジェクトのシミュレーションデータの一部を用いることとする [8]. このデータには次のデータが含まれている.

- 仮想マシン ID: 各仮想マシンに割り当てられた ID
- 時刻インデックス: シミュレーション内の時刻のインデックス

- CPU 使用率

このデータには一つの仮想マシンあたりに 288 のデータが存在し, これが 4394 ある. 本実験ではこの中から仮想マシンをいくつか選択して本実験に用いる.

5.2　リソース管理エージェントのシミュレーション

本論文の提案している手法を評価するにあたって, リソース管理エージェントのシミュレーションを実装した.

5.2.1　シミュレーション実装の構成

実装の全体像を図 5.1 に示す. js ファイルは main.js,detection.js,provision.js の 3 つに別れている. main.js で CPU 使用率の履歴をデータベースから取り出し,detection.js の getPrediction,getStability,getMargin を順に呼び出すことでマージンを計算する. マージンや CPU 使用率のデータを持ったホストマシンのデータである hosts を isOverload に渡すことで過負荷状態を検出する. isOverload は過負荷のホストマシンのリストと移行する仮想マシンのリストを返す. 次に provision.js の getRoom をホストマシンごとに呼び出しホストマシンの持つ余剰性能を計算する. さらに移行する仮想マシンのリストを findHost に渡して移行先ホストマシンを探し出す. sortHosts と sortVMs は仮想マシンとホストマシンをそれぞれサイズの大きい順, 余剰の大きい順に並べ替えるユーティリティ関数である.

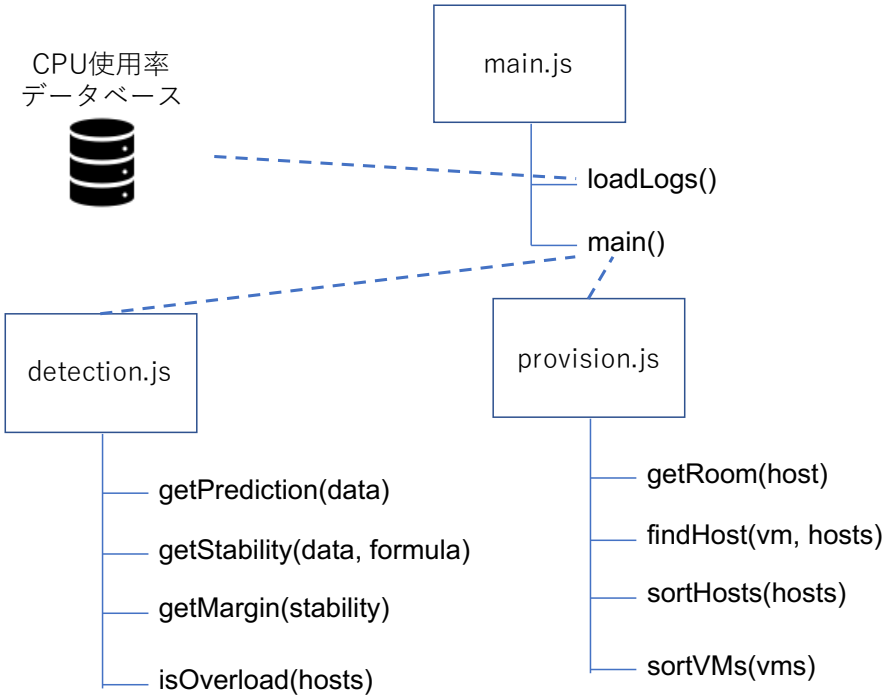

図 5.1: 実装のファイルと関数の構造

5.2.2　実験環境

実験環境は次のとおりである．また,CPU 使用率の履歴は膨大な数に上るため, ファイルから読み込むのではなく, データベースの一つである Mongodb を利用して扱った．本論文を実装するにあたり, メモリ使用量や実行速度については評価対象ではないため Node.js でも問題ないと判断した．

言語	Node.js (v10.14.2) + Typescript(v3.2.2)
PC	MacBook Pro (13-inch, 2017, Four Thunderbolt 3 Ports)
OS	macOS 10.14.2
CPU	3.1 GHz Intel Core i5
RAM	16 GB 2133 MHz LPDDR3
データベース	Mongodb(v4.0.3)

5.3　実験 1

本節では本論文の手法を評価する実験について説明する. 本実験の評価指標は先に述べたとおり「違反数」「マイグレーション数」,「ホスト数」の 4 つを用いる. 本実験では仮想マシンおよびホストマシンの CPU 動作周波数 × コア数 を「サイズ」として考えることとし, 全てこのサイズに基づいてホストマシンの容量や仮想マシンのリソース消費が計算される本実験における仮想マシンのサイズは 2400, 1700, 2000, 3000 のうちのいずれかとなっており, ホストマシンのサイズを変化させて実験を行った. ホストマシンのサイズは次の表 5.3 の通りであり, それぞれのサイズについて用いる仮想マシンを変えて 3 回ずつ実験を行った. また, 本研究の提案は主に固定値のマージンでは不足してしまう状況に対処するためのものであるから, 実験に用いる仮想マシンは CPU 使用率がほぼ 0%のままであるようなアイドル状態の仮想マシンではなく, 活発に CPU 使用率が変動しているデータの中から用いている. 選択の方法は CPU 使用率が高い順に仮想マシンのデータをソートし, 上位 1000 個の中からランダムに 100 個を選択して実験に利用している.

表 5.1: 実験の設定

(index)	ホストサイズ
0	3000
1	5000
2	8000
3	10000
4	12000

5.4　実験 1 の結果と考察

まず実験の結果のグラフを以下の図 5.2,5.3,5.4,5.5 に示す.

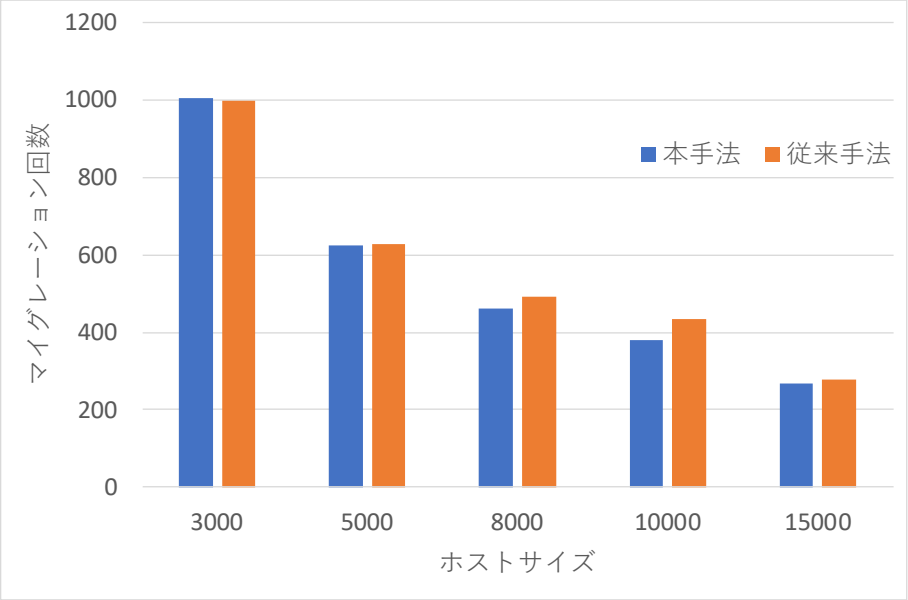

図 5.2: マイグレーション回数

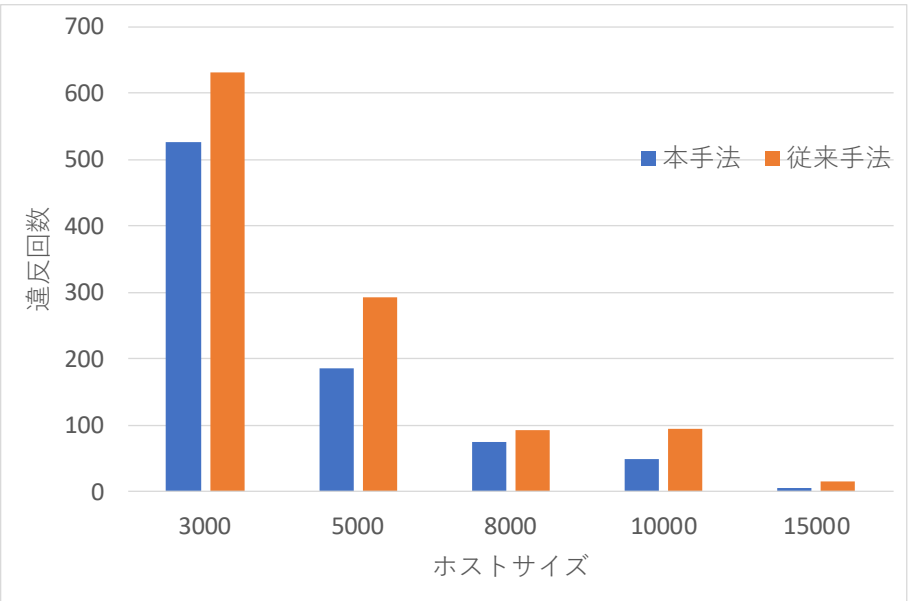

図 5.3: 違反回数

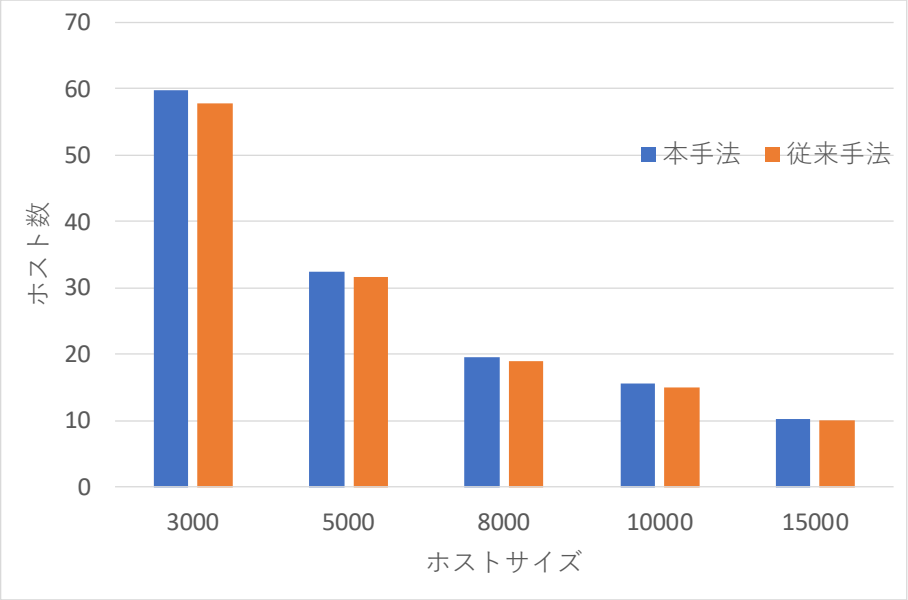

図 5.4: ホスト数

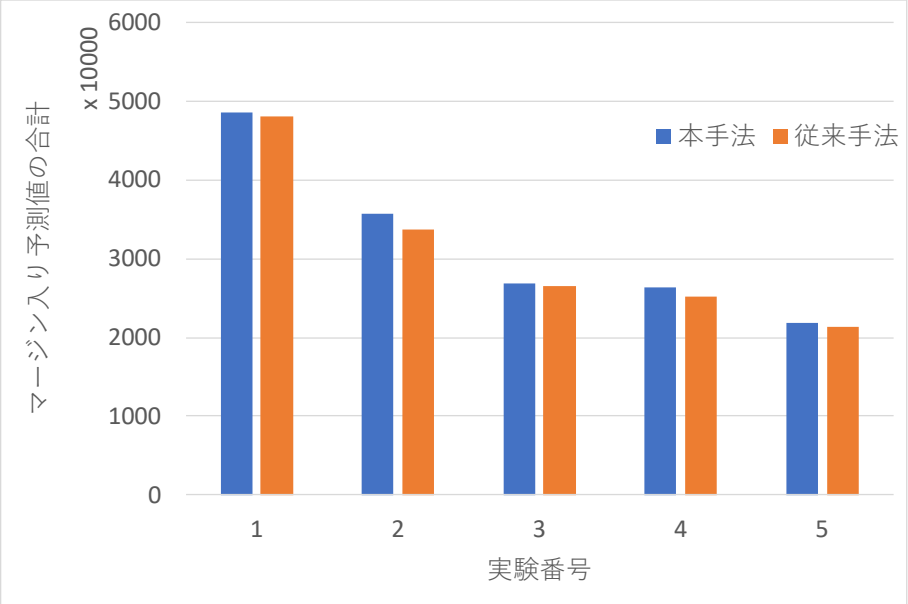

図 5.5: マージン入り予測値の合計

ホストサイズが小さいほど既存手法, 本手法共にマイグレーション回数,

違反回数, ホスト数は減っている. ホスト数が減っていくことは当たり前であるが, 特に違反回数の減少が顕著であり, 節 4.5.2 で説明したモデルに即した振る舞いをしていることが見て取れる. 既存手法と本手法を比較していくと, ホストサイズ 3000 の時マイグレーション数はほぼ変わらずホスト数は増加していて違反数は減少している. これはホスト数が増えて分散されたために違反数もそれに応じて減ったとも考えられる. ホストサイズが 5000 の時にマイグレーション数, ホスト数, 違反回数はほぼ変わらない結果となっている. ホストサイズが 7500,10000 の時にはマイグレーション数と違反回数が減少し, ホスト数は微増している. 特にホストサイズが 10000 の時にはマイグレーション回数は約 13%減少し, 違反数は約 49%も減少しており, 本手法が効果的にリソース利用の効率をあげていると言える. ホストサイズが 15000 になると効果は減少し違反回数はほぼ 0 となり, ホスト数はほぼ同数, マイグレーション回数も微減に留まっている.

また, マージン入り予測値の合計は全ての実験で従来手法に対して増加しているがその割合はわずかで, 安定度の低い仮想マシンには大きなマージンを付与し, 安定度の高い仮想マシンには小さなマージンを付与することで全体としてのリソース確保の量はほぼ変化していないことがわかる.

以上の結果から, 本手法は既存のマージンを固定した手法を比較してマイグレーション回数・ホスト数を減少もしくはほぼ同数に保ちつつ, 違反回数を最大で半数程度に減少させることが可能であると言える. また, 仮想マシンがホストマシンに対して数個程度配置される小さなホストサイズもしくは違反回数自体がほぼ 0 になってしまう程ホストサイズが大きい場合には本手法の大きい効果は得られないということも言える. このことからホストマシンのサイズが中程度であれば本手法は既存手法に比べて違反回数を減少させながらより効率的なリソース管理を行えることが言える. また, リソースの確保量が変わらないことから, 本手法がただマージンを大きく確保して違反回数等を改善しているわけではないことが言える. 本手法はパブリッククラウドに置いて提供コストを抑えたい場合や, プライベートクラウドでリソースが限られている状況に適していると言える.

5.5　実験 2

節 5.4 でも述べたとおり, 節 4.5.2 で説明したモデルに即した結果を示していることから, 追加提案しているオーバーコミット度合いを動的に増加

させる手法についても効果を検証するために実験 2 を行った. 実験 2 では節 4.4 同様に上限の値および係数を次の数式の $LIMIT$ および $DIVIDER$ を変動させて得られた結果から決定することとした.

$$coef = min(LIMIT, max(1.0, 1 + \frac{num\_of\_vms}{DIVIDER})) \tag{5.1}$$

$DIVIDER$ を大きくするということはオーバーコミット度合いの増加が緩やかになることを示し,$LIMIT$ を大きくすることはオーバコミット度合いの上限を上げることを示している. 係数の設定は次のように行った.0 番目の実験は比較のため, オーバーコミットの度合いは変化させない場合の実験である. 実験 2 は節 5.3 おいてもっとも優れた結果を示したホストマシンのサイズが 10000 の条件で行い, 次の表 5.5 に示す通りの条件に 100 個の仮想マシンを選択して実験を行った. 表 5.5 は $LIMIT$ および $DIVIDER$ の組み合わせを示しており, 図 5.6,5.7,5.8 の横軸と対応している.

表 5.2: 実験 2 の設定

LIMIT	DIVIDER
1	-
1.1	50
1.1	75
1.1	100
1.1	125
1.1	150
1.1	175
1.1	200
1.1	250
1.1	300
1.2	50
1.2	75
1.2	100
1.2	125
1.2	150
1.2	175
1.2	200
1.2	250
1.2	300
1.3	50
1.3	75
1.3	100
1.3	125
1.3	150
1.3	175
1.3	200
1.3	250
1.3	300

LIMIT	DIVIDER
1.4	50
1.4	75
1.4	100
1.4	125
1.4	150
1.4	175
1.4	200
1.4	250
1.4	300
1.5	50
1.5	75
1.5	100
1.5	125
1.5	150
1.5	175
1.5	200
1.5	250
1.5	300

5.6　実験 2 の結果と考察

実験 2 の結果のホスト数, マイグレーション数, 違反数をそれぞれ図 5.6,5.7,5.8 に示す. $LIMIT = 1$ の条件は, オーバーコミット度合いを変化させない場合の結果を表している. $LIMIT = 1.2$ までは $DIVIDER$ が増加するにしたがってホスト数, マイグレーション数が増加し, 違反数が減少しているが,$LIMIT$ が 1.3 以上になるとその傾向は不規則になっていて, マイグレーション回数は最大で 2 倍程度まで大きくなり, 違反回数は大幅に増加している場合もある. ホスト数に関して見ると, 当然の結果ではあるが $LIMIT$ に応じて減少傾向にある. 以上のことから, オーバーコミットの度合いを増加させる場合, ホストマシンのサイズの 1.2 倍程度までが安定して違反回数の増加を抑えてホストマシンの数を減らせると言える. マイグレーション回数や違反回数の増加を許容できる場合, それ以上にオーバーコミットすることでよりホストマシンの数を減らすことは可能である. SLA（Service Level Agreement）などのない研究室の小規模クラウドなどで限られたリソースを用いてプライベートクラウドを運用したい場合などにこの手法は用いることができると考えられる.

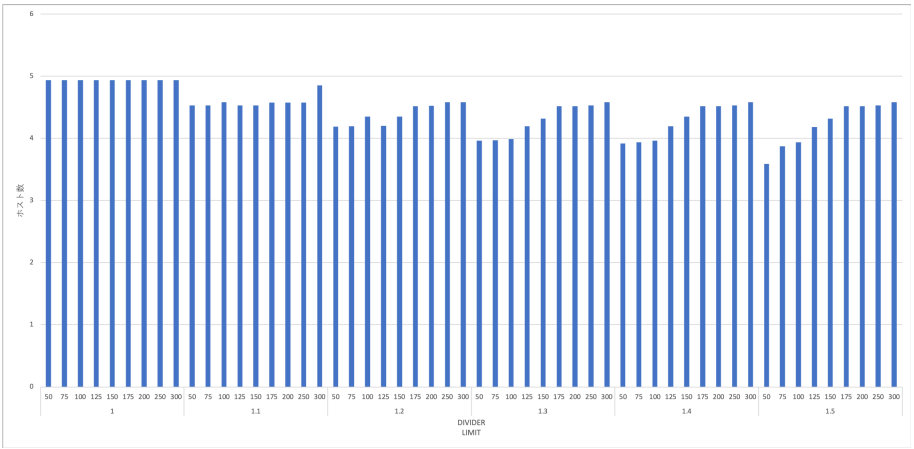

図 5.6: 実験 2 の結果: ホスト数

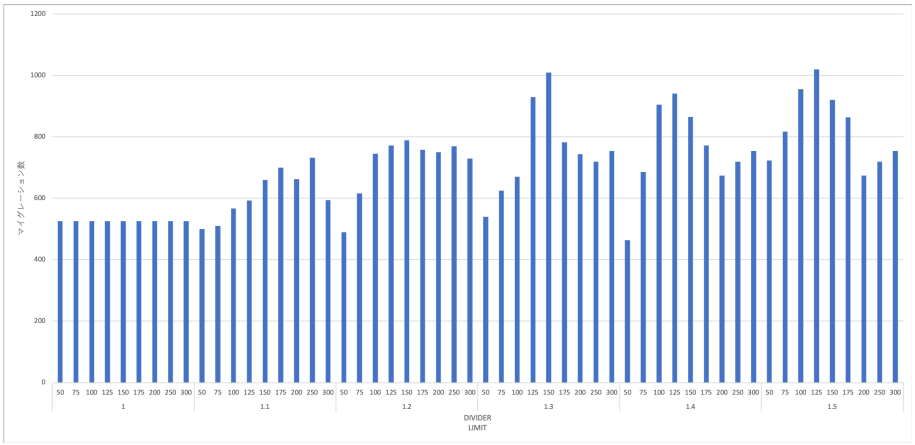

図 5.7: 実験 2 の結果: マイグレーション数

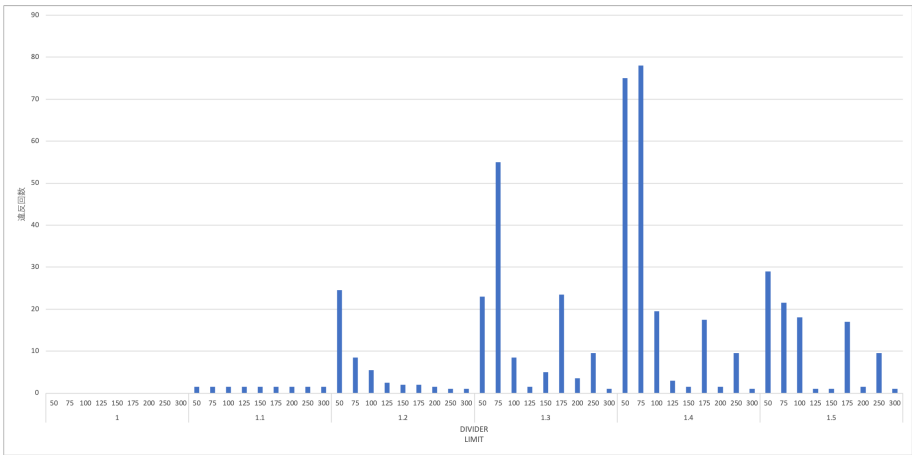

図 5.8: 実験 2 の結果: 違反数

第6章　おわりに

本研究ではクラウドコンピューティングという最近利用が急増しているコンピューティングリソースの利用について, 特に IaaS における仮想マシンの配置について焦点を当てた. 仮想マシンを大量に配置するクラウドではリソースの効率的管理を行うことによりオンプレミスに比べて大きくコストを削減できる可能性がある. リソースの効率的利用に用いられつ技術としてライブマイグレーション, オーバーコミットがあり, これらはリソースの利用予測に基づいて行われている.

CPU 使用率を予測に用いた既存の手法においてはライブマイグレーションの決定には予測値に固定されたマージンを付与した値を用いていたが,CPU 使用率は変動が大変大きいものである特性があった. 更に仮想マシン上で稼働しているアプリケーションごとに CPU 使用率の変動の仕方には特性があった.

これらの特徴に対して固定されたマージンでは対応しきれない場合があり, 本論文では仮想マシンの安定度に応じた動的マージン決定およびそれを用いたライブマイグレーション手法の提案を行った. 手法を評価する実験においては研究室や中小企業に想定される中程度のホストマシンのサイズにおいて特に違反回数を減少させ, リソース利用効率を高めることが確認された. 実験 2 においては, 違反回数の増加を許容する場合においては, オーバーコミットの割合をホストマシンに配置される仮想マシンの数に応じて変化させることでホスト数を減らすことができ, より限られたリソースで運用できる可能性が示された.

6.1　今後の課題・展望

今後の課題であるが, 本研究においてマージン決定の係数や実験 2 における係数について, ヒューリスティックな決定方法に基づいている. 実用性を高めるためにはこれらの部分を自動化する必要があると言える.

また, 本論文において CPU 使用率の予測には線形回帰を用いたが, 予測手法は他にも非線形回帰や隠れマルコフモデル, 機械学習などが存在する. 他の予測手法に対して本論文の提案である動的なマージン設定は有効であるのかを確かめることで実用性を確かめることができると考えられる.

マイグレーションやマージンの決定方法についても,CPU 使用率以外にもメモリやネットワークなど仮想マシンの稼働に関わる様々なパラメータが存在するため, それらを考慮することができればより効率のいいリソース管理エージェントを実現できると考えられる.

第7章　謝辞

研究活動全般において右も左もわからない自分に対して幅広い視野からの厳しい指導と激励をいただきました本学基幹理工学部深澤研究室教授深澤良彰先生には心より感謝の意を表します. また, 研究活動を行う上で，研究方針や外部活動において何から何まで直接的に広く指導を賜り, 論文執筆にあたっては人前に出して最低限恥ずかしくないような文章に仕立て上げて頂いた, 茨城大学助教高橋竜一先生にも心より感謝致します.

そして, 大学院生の方々など主に深澤研究室, 鷲崎研究室のメンバーには常に刺激的な議論を頂き, 精神的にも支えられました. 感謝いたします.

最後に，私の意志を尊重し，経済面と生活面において多大なる支援をしていただいた両親に深く感謝を致します.

第8章　研究業績

- 仮想マシンの CPU 使用率の負荷予測に対する安定度を用いた動的マージン設定手法
  坂本竜, 高橋竜一, 深澤良彰
  知能ソフトウェア研究会, 2018 年 2 月 18 日-3 月 2 日

- 仮想化環境における動的なマージン設定を用いたライブマイグレーション手法
  坂本竜, 高橋竜一, 深澤良彰
  合同エージェントワークショップ＆シンポジウム 2018 (JAWS2018), 2018 年 9 月 13-15 日

- **Stability-Based Live Migration Using Dynamic Margin**
  Ryu Sakamoto, Ryuichi Takahashi, Yoshiaki Fukazawa
  9th International Conference on Internet Technologies & Society 2019 (ITS2019), 2019 年 2 月 8-10 日

参考文献

[1] Alex Abrahao, Bruno; Zhang. Characterizing application workloads on cpu utilization for utility computing. Technical report, Hewlett-Packard Development Company. `http://www.hpl.hp.com/techreports/2004/HPL-2004-157.pdf`.

[2] Soramichi Akiyama, Takahiro Hirofuchi, and Shinichi Honiden. Evaluating Impact of Live Migration on Data Center Energy Saving. No. 25330097, 2014.

[3] F. Farahnakian, P. Liljeberg, and J. Plosila. Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 357–364, Sep. 2013.

[4] Fahimeh Farahnakian, Pasi Liljeberg, and Juha Plosila. LiRCUP: Linear regression based CPU usage prediction algorithm for live migration of virtual machines in data centers. *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*, pp. 357–364, 2013.

[5] Chaima Ghribi, Makhlouf Hadji, and Djamal Zeghlache. Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms. *Proceedings - 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013*, pp. 671–678, 2013.

[6] Google. Search engine strategies conference. `https://www.google.com/press/podium/ses2006.html`.

[7] Nguyen Trung Hieu, Mario Di Francesco, and Antti Yla-Jaaski. Virtual Machine Consolidation with Usage Prediction for Energy-Efficient Cloud Data Centers. *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, pp. 750–757, 2015.

[8] Vivek Pai KyoungSoo Park. Comon project, 2012.

[9] C. Li, D. Feng, Y. Hua, W. Xia, L. Qin, Y. Huang, and Y. Zhou. Bac: Bandwidth-aware compression for efficient live migration of virtual machines. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, May 2017.

[10] V. D. Maio, G. Kecskemeti, and R. Prodan. An improved model for live migration in data centre simulators. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 527–530, May 2016.

[11] S. B. Melhem, A. Agarwal, N. Goel, and M. Zaman. A markov-based prediction model for host load detection in live vm migration. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 32–38, Aug 2017.

[12] NIFCLOUD. クラウドを支える基盤技術「仮想化」とは？ `https://cloud.nifty.com/navi/beginner/virtualization.htm`.

[13] J. P. Putra, S. M. S. Nugroho, and I. Pratomo. Live migration based on cloud computing to increase load balancing. In *2017 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pp. 286–290, Aug 2017.

[14] Z. Wu, X. Li, P. Garraghan, X. Jiang, K. Ye, and A. Y. Zomaya. Virtual machine level temperature profiling and prediction in cloud datacenters. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pp. 735–736, June 2016.

[15] クラウドエース. Iaas paas saas の違い. `https://www.cloud-ace.jp/report/detail01/`.

[16] 深澤良彰坂本竜. 仮想マシンの cpu 使用率の負荷予測に対する安定度を用いた動的マージン設定手法. 信学技報, 第 117 巻, pp. 49–54, Mar. 2018.